Assignment 1 – COEN 346

Deadline: Sunday February 15 by 23:59

Weight: 8%

**Submission instructions:**

- Copy all Java files and your report file in a folder and put your student id(s)_(assignment number) as the folder name.
- Zip the folder and Submit it via Moodle.
- If you are working in a pair, name the ZIP file using the student IDs of both students. Only one member of the group needs to submit the assignment.

# Multithreaded Zero-Day Attack

## 1- Introduction

Vulnerabilities in software can be discovered by hackers, security experts, or even users. In such cases, software must be patched as soon as possible to protect users from cybercrimes (e.g., stealing sensitive information, shutting systems down, etc.). Another scenario can also occur: crackers may keep these vulnerabilities hidden in order to sell them on black markets on the dark web. These vulnerabilities are purchased to design cyberattacks that are launched at specific dates and times with carefully planned strategies. As a result, security experts may not have enough time to protect the system. This type of attack is known as a zero-day attack. If the only method to avoid zero-day attacks were to wait for the software vendor to patch the program, users would be waiting a long time. While there is no single "magic bullet" solution to protect a network from all zero-day vulnerabilities, there are certain measures that individuals and organizations can take to improve security, such as strong anti-spam and antivirus protections, although these solutions may increase costs.

## 2- Proposed Solution

Speeding up the process of searching for system vulnerabilities can reduce risk. In this context, it is necessary to study system behavior. One of the most efficient methods for automatically analyzing system behavior is logging. Logging is a common programming practice that involves adding statements to source code in order to record critical runtime information. The numerous uses of logs in software system management activities—such as anomaly detection, fault debugging, performance diagnostics, workload modeling, and system behavior analysis—demonstrate the importance of logging. However, it is impractical to manually read and analyze logs to extract such complex information. Therefore, based on your experience in the course COEN 346, your task is to design and implement a parallel solution that processes log files using multithreading. You are required to implement a Java program using a multi-level threading model. This assignment involves two types of threads (Master and Worker), as explained in the following sections. The main objective is to search for specific vulnerability patterns in a given dataset.

The dataset consists of a single text file containing artificial logs. Each line in the file represents a single log entry with the following attributes:

- Date and Time of task.
- Service Id
- Log statement with a length of 265 bytes

Here is an example of one line of the dataset file:

VM1 20210907_ 080513 AM Services_3757 Log:**ZY8BXAW5P2QZU9Q5P01PXNEI3PZ…….**

You need to use the provided Java file (LevenshteinDistance.java), which contains the Levenshtein distance algorithm.

This algorithm helps search log files for possible vulnerability patterns. The Java class contains three functions and a Boolean attribute called ***acceptable_change***:

- **Calculate()** function: This function takes two string parameters with the same length and returns their differences as an integer.
- **Measure_Change_Ratio() :** This function measures the change ratio between the compared strings. If the similarity ratio between two strings is equal or bigger than 0.05 it sets the ***acceptable_change*** attribute to true.
- **isAcceptable_change():** This function returns the value of ***acceptable_change*** attribute

## 2-1 Master Thread

The master thread is your first launched thread (by main thread) that reads the text file and loads its content into an array of strings. Each item of this array contains one line from the file. This task should be done on the class constructor handling IO exception if the file does not exist.

The Master Thread class has five attributes as follows:

- **Vulnerability Pattern**: his attribute stores the vulnerability pattern that needs to be searched for.
- **worker_number**: This is the number of running threads under the master thread. The initial value for this attribute is 2.
- **Count**: This represents the number of running threads under the master thread. The initial value of this attribute is 2.
- **Avg**: This stores the initial value of approximate_avg which is 0.
- **approximate_avg**: This represents the average number of detected vulnerabilities (Count / number of lines in the file) and is used to determine whether to increase the number of launched worker threads. If the difference between **Avg** and **approximate_avg** is greater than 20% (0.2), the master thread increases the number of worker threads (**worker_number**) by 2 for the next iteration and updates **Avg** to the new value of **approximate_avg**.

The master thread operates in an iterative manner. It begins by launching two worker threads under its control and, starting from the beginning of the array, assigns one line (one array element) to each thread. Each worker thread searches its assigned log entry for the vulnerability pattern, and if the pattern is found (i.e., if acceptable_change is true), it increments the value of Count by 1. More details about the worker threads are provided in the next section.

After the worker threads finish their tasks, the master thread calculates the updated value of approximate_avg based on the new value of Count. If this value is at least 20% greater than the value of Avg for the current number of launched threads, the master thread first sleeps for 2000 milliseconds, then updates Avg with the newly calculated approximate_avg, and increases the number of worker threads by 2 for the next iteration (e.g., from 2 to 4), and so on. Otherwise, it launches the same number of worker threads in the next iteration.

Please note that this sleep period helps you observe the transition points at which the number of launched worker threads increases by 2.

## 2-2 Worker Thread

The task of each worker thread is to search its assigned log statement for possible similarity with the **Vulnerability pattern.** To do so, the worker thread starts from the beginning of the assigned log statement and recursively passes substrings, each with a length equal to that of the vulnerability pattern, to the Levenshtein Java functions. This process is used to search the log statement for the vulnerability pattern. If, at the end of the search process, the value of **acceptable_change** is true, the worker thread must increment the number of detected vulnerabilities (**Count**) in the master thread.

You can consider **Vulnerability Pattern** = **V04K4B63CL5BK0B** in your implementation. Here are two examples of successful and unsuccessful search processes for the worker thread:

| | | |
|---|---|---|
| Vulnerability pattern | V04K4B63CL5BK0B | |
| Log statement | YRKV04K4B63CL5BK0BOZJ | |
| **Run 1** | V04K4B63CL5BK0B<br>YRKV04K4B63CL5BK0BOZJ | |
| **Run 2** | V04K4B63CL5BK0B<br>YRKV04K4B63CL5BK0BOZJ | |
| **Run 3** | V04K4B63CL5BK0B<br>YRKV04K4B63CL5BK0BOZJ | ✓<br>`isAcceptable_change() function return true` |

---

| | | |
|---|---|---|
| Vulnerability pattern | V04K4B63CL5BK0B | |
| Log statement | RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 1** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 2** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 3** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 4** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 5** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 6** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | |
| **Run 7** | V04K4B63CL5BK0B<br>RN5NM1WUFR7DV0LAOAOB8 | `isAcceptable_change() function return false`<br>✗ |

## 3- Supported Files

You will have two package folders provided to you as follows:

1- **Support:** This folder contains **LevenshteinDistance.java** means the implementation of text similarity algorithm for two strings of the same length is already provided to you.
2- **DataSet:** This folder contains **vm 1.txt** which is the dataset of VM logs.

## 4- Submitted Documents

This assignment can be completed in groups of two students. The deliverable consists of well-commented code and a report of at least one page that provides a high-level description of the code, including the methods/functions/threads used and the overall program flow. You must also explain the synchronization method you used and discuss the rationale behind your choice.

This assignment is worth 8% of your total programming assignment mark. Of this, 80% is allocated to the code and 20% to the report.